Privacy-preserving RFID Authentication based on Cryptographical Encoding

Tao Li Wen Luo Zhen Mo Shigang Chen
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL, USA
Email: {tali, wluo, zmo, sgchen}@cise.ufl.edu

***Abstract*—Radio Frequency IDentification (RFID) technology has been adopted in many applications, such as inventory control, object tracking, theft prevention, and supply chain management. Privacy-preserving authentication in RFID systems is a very important problem. Existing protocols employ tree structures to achieve fast authentication. We observe that these protocols require a tag to transmit a large amount of data in each authentication, which costs significant bandwidth and energy overhead. Current protocols also impose heavy computational demand on the RFID reader. To address these issues, we design two privacy-preserving protocols based on a new technique called *cryptographical encoding*, which significantly reduces both authentication data transmitted by each tag and computation overhead incurred at the reader. Our analysis shows that the new protocols are able to reduce authentication data by more than an order of magnitude and reduce computational demand by about an order of magnitude, when comparing with the best existing protocol.**

I. Introduction

Radio Frequency IDentification (RFID) technology has been widely used in many applications, such as inventory control, object tracking, theft prevention, and supply chain management [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. A RFID system consists of a number of tags and a reader. Each tag is attached to an object, which can be a product in a warehouse, a merchandize in a retail store, an animal in a zoo, or a piece of medical equipment in a hospital. The tag has an integrated antenna for receiving and transmitting radio-frequency (RF) signal. By communicating with tags via RF signals, the reader is able to obtain the ID or certain properties of each object or collect aggregated information about a group of objects.

Privacy-preserving authentication is a very important problem. In secure RFID systems, a reader will accept a tag's information only after it authenticates the tag. A tag may be attached to a pharmaceutical product that carries patient information, a passport that carries a person's identification, or a commercial product that carries information about manufactured date, expiring date, ingredients, etc. Some applications require *privacy-preserving* authentication, in which a tag should not give out any identifying information during authentication process. Suppose a police tries to use a mobile RFID reader to authenticate a driver's license embedded with a RFID tag, and the reader has access to a database of all secret keys that are pre-installed in driver licenses. The reader has to know which key it should use to perform authentication. In a typical authentication protocol, the license needs to transmit an identifying number to the reader, which will use that number to search the database for the right key. However, this leads to a security loophole. The identifying number, transmitted wirelessly, reveals the presence of the carrier. Fake readers may initiate the authentication process at chosen locations and chosen times. They will terminate the process after the identifying number is received. It allows them to monitor the whereabout of the license's carrier.

Can we design an authentication protocol that allows a RFID reader to authenticate a tag without requiring the tag to transmit any identifying data? Ideally, the information transmitted from a tag should look totally different and random each time the tag is authenticated. Weis et al. [11] have designed a privacy-preserving authentication protocol, in which the reader has to try all keys in the database in order to see if there exists one that matches the authentication data sent from the tag. The computation complexity of this protocol for authenticating a single tag is $O(N)$, where N is the total number of tags, e.g., the number of all driver licences in a state for the example in the previous paragraph. In order to accelerate the authentication, other existing protocols [12], [13], [14], [15] organize all keys in a tree structure. They are able to reduce the computation overhead to $O(\sigma d)$, where σ is the degree of the tree, i.e., the number of child nodes that each internal node has; and d is the depth of the tree. Typically, d is a multiple of $\log N$. The problem of these protocols is that each tag has to transmit $O(d)$ cryptographic hash values. Suppose the depth of the tree is 30. Each hash value is 160 bits if SHA-1 is used. The tree-based protocols require a tag under authentication to transmit $30 \times 160 = 4,800$ bits, which is a lot in the context of RFID systems. For battery-powered active tags, energy consumed by a large amount of transmission shortens the lifetime of the tags.

Another drawback of the current tree-based protocols is their high computational demand on the RFID reader. For each authentication, the reader (or the authentication server that is connected to the reader) has to compute $\frac{d\sigma}{2}$ cryptographic hashes. When $d = 30$ and $\sigma = 16$ [15], the reader computes 240 hashes for authenticating a single tag. This can be a problem when the reader has to authenticate numerous tags, for instance, in a large warehouse. In addition,

978-1-4673-0775-8/12/$31.00 ©2012 IEEE

high computation overhead at the server makes it vulnerable to denial-of-service attacks, where fake tags continuously prompt for authentication, preventing the reader from authenticating real tags. For example, consider a highway toll booth that authenticates vehicles wirelessly for automatic payment through RFID passes. An adversary may plant a device nearby, which continuously replies to the queries from the booth's reader and pretends that a large number of tags are waiting to be authenticated, causing a denial-of-service attack. Reducing computation overhead at the reader can help alleviate such a problem.

In this paper, we propose a new technique called *cryptographical encoding*, which compresses the authentication data from a tag to a small fraction of its original size, without revealing any identifying information. Based on the idea of cryptographical encoding, we design two privacy-preserving authentication protocols. Not only do they drastically reduce the amount of authentication data, but also they cut the overhead at the reader from $\frac{(\sigma+1)d}{2}$ hash computations to $d+1$ computations. By reducing the amount of authentication data transmitted from each tag, we reduce the tags' energy consumption as well as the authentication time, considering that RFID systems operate at very low transmission rates. By reducing the reader's computation load, we make the authentication function more scalable for large RFID systems.

The rest of the paper is organized as follows. Section II defines the problem and system models in this study. Section III discusses the preliminary work. Section IV-V present our two protocols. Section VI draws the conclusion.

II. Problem Statement

Consider a large RFID system, where each tag is pre-installed a secret key and a reader has access to a database that contains the keys of all tags. We assume high-end tags that are capable of performing cryptographic hash and encryption. These tags are likely to have batteries to power their relatively rich on-tag resources. Communications between a reader and tags are time-slotted. The reader initiates the authentication process by transmitting a request message. The tag responds with certain authentication data, based on which the reader figures out which key from the database to use. Once the reader finds the right key that is pre-installed in the tag, any classical secret-key authentication protocol can be used for mutual authentication between the reader and the tag.

The reader itself may not be able to store a large number of keys and perform high-performance computations. In that case, it should be connected to a back-end server that is responsible for information storage and processing. We assume that the reader and the server are connected via a high-speed network. To simplify our discussion, we logically treat the reader and the server as one entity, still called *the reader*.

For the purpose of preserving privacy, neither a reader nor a tag should transmit any information that may reveal the tag's identify. More specifically, our requirement is that authentication data must look random each time a tag is authenticated. The tag (or reader) should not transmit any fixed number — which may serve for an identifying purpose — between any two authentications. Our goal is to reduce the amount of authentication data and computation for meeting the above requirement. The thread model assumes that an adversary may eavesdrop on any data exchanged between tags and reader(s).

The above requirement makes sure that an eavesdropper will not be able to determine whether two authentications are performed on the same tag or different tags. Without this requirement, it will not be the case. For example, in the previous driver's licence example, if a licence transmits a fixed number, a fake reader planted by the garage at the house of a driver will learn that number, and other fake readers planted at certain locations will learn when the driver passes these locations. But if authentication data look totally random, even under a threat model where an adversary is able to gather all authentication data communicated between tags and the reader, the adversary will not be able to associate a tag with a particular authentication.

III. Preliminary

Existing RFID authentication protocols can be generally divided into two categories: *non-tree-based* and *tree-based*.

A. Non-tree-based Protocols

Weis et al. [11] propose a privacy-preserving RFID authentication protocol, Hash Lock. In this protocol, the RFID reader generates a random number r and sends the number to a tag as an authentication request. After receiving r, the tag performs a hash operation $H(r, id)$, and transmits the hash value back to the reader, where H is a cryptographic hash function and id is the tag's identification number. The reader searches its database for the tag whose identification, together with r, can generate the same hash value, $H(r, id)$. If such a tag exists, the authentication is successful; otherwise, it fails. Here, id itself is used as the authentication key.

In Hash Lock, the tag only transmits a small amount of authentication data — one hash value — to the reader. However, it has a serious efficiency problem. The reader has to search its entire database and perform $O(N)$ hash computations on different identification numbers, where N is the total number of tags in the system.

Yao et al. [12] design a Random Walk based authentication Protocol (RWP). The protocol achieves $O(1)$ authentication overhead. It employs a reversible hash function, CuckooHash [16], such that the reader is able to recover a key k from its hash value $H(r, k)$ in $O(1)$ time. However, we know that cryptographic hash functions are not reversible. CuckooHash is not a cryptographic hash, and therefore is vulnerable when used in an authentication protocol. While it allows the RFID reader to recover a key from a hash value, it also makes an adversary's brute-force guessing attack easier to perform. Furthermore, in order to assign a proper hash value for each key so that the key can be recovered in constant time, the

reader has to allocate a very large amount of memory to keep the mapping information.

Lu et al. [17] propose a weak privacy model, where a tag transmits a key index to the reader so that the correct key can be located in $O(1)$ time. After authentication, there exists a secure communication channel protected by the common key between the reader and the tag. Using this channel, the reader uploads a new key and its corresponding new index to the tag for the next authentication. The purpose is to make the index information transmitted by the tag different. The problem is that, between two authentications by the real reader, the tag will always respond to fake readers with the same key index, which serves as the identifying information and breaks the privacy requirement. This can be a serious problem in applications where significant time gap may exist between two authentications by the real reader. For example, a driver's license may only be occasionally accessed wirelessly by polices due to speeding. Between such police-access events, the license will transmit the same key index to anyone that exploits the authentication protocol to automatically locate the license's carrier through fake readers planted at certain locations.

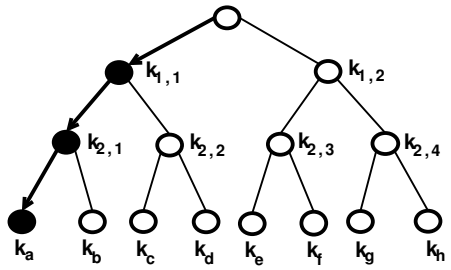


Fig. 1. Illustration for the balanced tree based protocol [13].

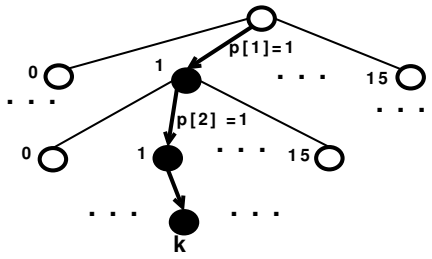


Fig. 2. Illustration for the ACTION protocol [15].

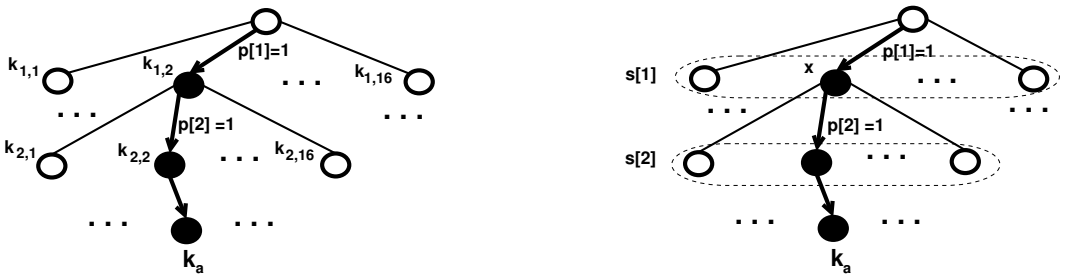
B. Tree-based Protocols

Dimitriou [13] proposes a balanced tree based protocol that is able to decrease the reader's computation overhead for authentication from $O(N)$ to $O(\sigma d)$, where σ is the degree of the tree (i.e., number of child nodes that each in-tree node has) and d is the depth of the tree, which is typically a multiple of $\log N$. As illustrated by a simple example of eight tags a through h in Figure 1, a balanced tree is constructed, where the secret keys, k_a through k_h, of tags are placed at the leaf nodes, and additional auxiliary keys, $k_{1,1}$, $k_{1,2}$, ..., are placed at the internal nodes. Without losing generality, let's consider tag a. All keys on the path from the root to the leaf node k_a are pre-installed in tag a; they are $k_{1,1}$, $k_{2,1}$, and k_a. The reader has access to the balanced tree during authentication. To begin with, the reader sends a random number r_1 to tag a. The tag chooses another random number r_2, and let r be the concatenation of the two random numbers. It then transmits d hash values, $v_1 = H(r, k_{1,1})$, $v_2 = H(r, k_{2,1})$, ..., together with r. Upon receiving these hash values, the reader uses the keys on the first level of the tree to compute hash values, $H(r, k_{1,1})$ and $H(r, k_{1,2})$, and see which one matches v_1. It follows the link of the matching key to the next level, where v_2 is used in a similar way to find the link down one level further. Hence, the hash values, v_1, ..., v_d, serve as indexing information for the reader to navigate in the tree until reaching the secret key k_a. Because each authentication has a different value of r, the hash values from a tag to the reader will be different even when the same keys are used.

The above protocol is vulnerable to the compromising attack [14], [15], which is able to partially break the privacy in a large RFID system by only compromising a small portion of the tags. The strong and lightweight RFID privacy authentication protocol (SPA) [14] moves a tag's secret key to a different leaf node in the tree and updates the key material of a tag after each authentication. However, it can only alleviate the compromising attack problem due to the dependency of key materials in the tree.

Lu et al. adopt a sparse tree in their ACTION protocol [15] in order to reduce the dependency among key materials. Both the degree and the depth of their sparse tree can be set large. ACTION randomly assigns a path indicator p to each tag. The path indicator is a binary string that is divided into d segments of length $\log_2 \sigma$, denoted as $p[1], p[2], ..., p[d]$. As illustrated in Figure 2, these segments serve as the *link indices* for the reader to locate the leaf node that carries the secret key k of the tag. The path indicator and the key are pre-installed in the tag. During authentication, in order to protect privacy, the tag cannot directly transmit its path indicator, a fixed value that can serve as an identifier. Instead, it transmits d hash values, $H(r, p[1])$, ..., $H(r, p[d])$. Upon receiving these hash values, the reader can easily figure out $p[1]$ from $H(r, p[1])$, $p[2]$ from $H(r, p[2])$, ..., $p[d]$ from $H(r, p[d])$. It simply computes $H(r, 0)$, $H(r, 1)$, ... $H(r, 15)$ and sees which one matches $H(r, p[1])$. For example, if $H(r, 5)$ matches $H(r, p[1])$, then $p[1]$ must be 5. In a similar way, the reader can find $p[2]$, ..., $p[d]$.

The key problem of ACTION is that any eavesdropper can do the same thing and figure out the path indicator and break the privacy. The size of the secret information $p[i]$, $1 \leq i \leq d$, is too short, and the number of its possible values is too few (e.g., 16) to be secure. After an eavesdropper captures $H(r, p[i])$, it is trivial for it to figure out $p[i]$: Simply trying out all possible values of $p[i]$ to see which one matches the



(a) Dimitriou's Protocol (b) CNP

Fig. 3. Per-node keys v.s. per-group keys.

received hash, just as what the reader does.

IV. Compact Navigation Protocol

In this section, we propose a *compact navigation protocol* (CNP) for privacy-preserving authentication in RFID systems. CNP is not only secure but also much more efficient than the existing protocols, thanks to a new technique called *cryptographical encoding*. We will present an enhanced version of the protocol in the next section.

A. Motivation

We also use a sparse tree, which means the degree σ and the depth d of the tree are large. A sparse tree is not suitable for Dimitriou's protocol because a tag has to transmit $O(d)$ hash values and the RFID reader has to perform $O(\sigma d)$ hash computations. As shown in Figure 3 (a), in order for the reader to figure out the first link to follow from the root of the tree, a tag a needs to transmit $H(r, k_{1,2})$ in Dimitriou's protocol. For ACTION, it would transmit $H(r, p[1])$. The purpose of these two approaches are the same: they help the reader figure out the index of the first link, $p[1]$. In this example, $p[1] = 1$. Similarly, the purpose for tag a to transmit $H(r, k_{2,2})$ in Dimitriou's protocol or $H(r, p[2])$ in ACTION is to help the reader figure out the index of the next link, $p[2]$, which happens to be 1 again in this example.

The problem is that ACTION is not secure, whereas Dimitriou's protocol has too much overhead. We propose a new technique, called *cryptographical encoding*, which is both secure and efficient. Instead of assigning a key to each node in the tree, we assign a key to a group of nodes. More specifically, we treat the children of each node as a group, and assign the group a key. Consider the 16 children of the root in Figure 3 (a). Dimitriou's protocol assigns one key to every child, and there are in total 16 keys, $k_{1,1}$, ..., $k_{1,16}$ for these children. Our approach is very different. As shown in Figure 3 (b), we assign one key, $s[1]$, to all children of the root as a group. We also assign a separate key to the children group of each internal node. For example, $s[2]$ is assigned to the children of x. These keys are called *group keys*.

Consider an arbitrary tag a. Let k_a be its secret key. We also use k_a to represent the leaf node that carries this key. Let P_a be the path from the root to k_a. Tag a is pre-configured with the following credential: 1) secret key k_a, 2) link indices $p[1]$ through $p[d]$, which together specify P_a, and 3) group keys $s[1]$ through $s[d]$ along the path. We stress that $s[i]$, $1 \leq i \leq d$, is not assigned to any single node but instead to the children group of the ith node on P_a.

During authentication, the tag computes $H(r, s[i])$, $1 \leq i \leq d$. For each $H(r, s[i])$, instead of sending the whole hash value (which may be 160 bits long), the tag only selects a few bits based on $p[i]$ and transmits those few bits to the reader. The method of bit selection in Section IV-D makes sure that the reader can figure out the value of $p[i]$ from the received bits. Essentially, those few bits of $H(r, s[i])$ encode $p[i]$ cryptographically.

In summary, instead of sending $O(d)$ hash values, a tag sends a few bits from each hash value as authentication data. Instead of computing σ hashes in order to figure out each link index $p[i]$, the reader only computes one hash in most cases.

B. Overview

Our compact navigation protocol (CNP) for privacy-preserving authentication consists of three phases: *initialization phase, encoding phase*, and *decoding phase*. The initialization phase constructs a sparse tree of key materials. We assign a path navigator p and a secret key k to each tag in the RFID system. Based on the path navigator, we insert a tree branch that connects to a leaf node storing k. A separate group key is assigned to the children group of every node on the branch. Each tag is also configured with its key materials. In the encoding phase, a tag generates a set of authentication data based on its key materials. It then sends the authentication data to the reader. In the decoding phase, the reader authenticates the tag based on received data. Data from an authentic tag will successfully guide the reader to traverse in the tree from the root to the correct leaf node; data from a counterfeit tag will be detected and rejected during the traversal process.

C. Initialization Phase

Initially the sparse tree is set to be empty. Let r be the root node of the tree. Whenever a new tag a is added to the system, we expand the tree by adding the key materials of

tag a into the tree. We first assign a path navigator p and a secret key k to the tag. We then divide p into d link indices of length $\log_2 \sigma$, which are denoted as $p[1], p[2], ..., p[d]$. Let '||' be the concatenation operation, and $p = p[1]||p[2]||...||p[d]$. We construct a tree branch based on these link indices and insert this branch into the tree. First, we insert the children of the root into the tree and add a key $s[1]$ for this children group if the children are not already in the tree. Second, we move to the $p[1]$th child of the root. Let's denote this node as x. We insert all children of node x into the tree and add a key $s[2]$ for this children group if the children are not already in the tree. Third, we move to the $p[2]$th child of node x. We repeat the above step to add children and group keys until we reach a leaf node after following the last link index $p[d]$. The leaf node will be assigned the secret key k of the tag.

Before the tag is deployed, it must be pre-configured with the key materials, including its secret key k, the path indices, $p[1]$, ..., $p[d]$, and the group keys, $s[1]$, ..., $s[d]$.

The initialization phase sets the keys that are needed when the reader authenticates a tag. The actual authentication process involves the next two phases: First, the tag computes authentication data in the encoding phase. Then, the reader completes authentication through the decoding phase.

D. Encoding Phase

Before a reader and a tag can perform mutual authentication, they must identify a common secret key, i.e., the key k that is pre-installed in the tag. To inform the reader about which key to use, the simplest approach is for the tag to transmit the link indices, $p[1]$, ..., $p[d]$. The reader can use these indices to traverse the tree to a leaf node carrying k. However, the link indices are fixed values that can also be used to identify the presence of a tag. In the following, we explain our cryptographical encoding method that encode $p[1]$, ..., $p[d]$ in compact authentication data based on the group keys $s[1]$, ..., $s[d]$.

To begin the authentication process, the reader generates a random number r_1 and sends it to the tag as an authentication request. After receiving r_1, the tag also generates a random number r_2 and XORs it with r_1. The result is denoted as r. This number is different for each authentication. The tag then computes $H(r, s[i])$, $1 \leq i \leq d$, and selects a few bits from $H(r, s[i])$ to encode $p[i]$. Below we only explain how to selects bits from $H(r, s[1])$ to encode $p[1]$. Other $p[i]$ values can be similarly encoded.

The reader splits $H(r, s[1])$ into σ segments, denoted as $v[0]$, $v[1]$,...,$v[\sigma - 1]$. If the hash value is 160 bits long, each segment has $\frac{160}{\sigma}$ bits; when σ is 16, each segment has 10 bits. We may simply use $v[p[1]]$ to encode $p[1]$. For example, if $p[1] = 0$, the tag will transmit $v[0]$ to the reader; if $p[1] = 1$, the tag will transmit $v[1]$ to the reader; and so on. When the reader receives these bits, it also computes $H(r, s[1])$ and divides the result into σ segments. The reader compares the received bits with these segments to find the matching one. If the received bits match the ith segment, the reader traverses to the ith child of the root in the tree.

The above approach works only when $v[p[1]]$ is not equal to any other segment $v[i]$, $i \neq p[1]$. This is true with high probability, 98.5% if $\sigma = 16$ and each segment is 10 bits long. However, if the tag finds that $v[p[1]]$ is not unique i.e., $v[p[1]]$ is equal to another segment, it has to expand $v[0]$, $v[1]$, ..., $v[\sigma - 1]$ with more bits, such that $v[p[1]]$ becomes unique. To do so, the tag sets $r' = H(r, s[1])$ and computes another hash $H(r', s[1])$. It divides the hash result into σ segments, denoted as $v'[0]$, $v'[1]$, ..., $v'[\sigma - 1]$, and appends these segments to $v[0]$, $v[1]$, ..., $v[\sigma - 1]$, respectively. After that, if $v[p[1]]$ is still not unique, the tag repeats the above procedure for more bits: set $r'' = H(r', s[1])$, compute $H(r'', s[1])$, divide the hash result, ..., until $v[p[1]]$ becomes unique.

In most cases, however, we do not need to add additional bits to $v[p[1]]$ in order to make it unique. In fact, we can often remove bits from $v[p[1]]$ and still keep it unique. The tag only needs to transmit the shortest prefix of $v[p[1]]$ that is different from the prefixes of other segments. For example, suppose $p[1] = 1$, $\sigma = 4$, and $H(r, s[1]) =$ "0000||0101||1011||1100". We know that $v[p[1]] =$ "0101" and it is unique. Its two-bit prefix "01" is also unique because the two-bit prefixes of other segments are "00", "10", and "11", respectively. Hence, the tag only needs to transmit "01", instead of the entire $v[p[1]]$. In summary, the tag uses the *shortest unique prefix* of $v[p[1]]$, denoted as f_1, to encode $p[1]$.

Similarly, the tag computes f_i to encode $p[i]$ based on group key $s[i]$, $i > 1$. The authentication data that the tag transmits to the reader are $\{r_2, f_1, f_2, ..., f_d, H(r, k)\}$.

E. Decoding Phase

After the reader receives the authentication data $\{r_2, f_1, f_2, ..., f_d, H(r, k)\}$ from the tag, it first reproduces r by XORing r_2 with r_1. The reader then decodes f_i, $1 \leq i \leq d$, to recover the link indices $p[i]$, based on which it can traverse the tree to find the leaf node carrying the secret key k of the tag. Below we only explain how to decode f_1 for $p[1]$. Other $p[i]$ values can be similarly derived.

To decode f_1, the reader computes the hash value $H(r, s[1])$ and divides it into σ segments, $v[0]$, $v[1]$, ..., $v[\sigma - 1]$. If the length of f_1 is equal to or smaller than the segment size, the reader tries to find a segment that matches f_1 as a prefix. For an authentic tag, according to the design of the encoding phase, there should be one and only one segment $v[j]$ that matches f_1 as a prefix. In that case, $p[1] = j$. On the other hand, if the reader cannot find any segment or it finds multiple segments that match f_1 as a prefix, it knows that the tag is a counterfeit one.

If f_1 is longer than the segment size, the reader needs to add more bits to the segments by performing additional hashes: set $r' = H(r, s[1])$, compute $H(r', s[1])$, divide the hash result into σ segments, append them to $v[0]$, ..., $v[d]$, respective, ..., until the segment size becomes equal to or larger than the length of f_1. After that, the reader tries to find a segment $v[j]$ that matches f_1 as a prefix. If it is successful, it will set $p[1] = j$.

After the reader decodes f_i, $1 \leq i \leq d$, and finds all path indices, it uses these indices to traverse the tree and reach a leaf node that carries a key k'. It performs a keyed hash $H(r, k')$, and compares it with the received $H(r, k)$. If they are the same, the reader knows that $k' = k$ and the tag is authentic. To allow the tag to authenticate the reader, the reader computes $H(r_2, k)$ and transmits it to the tag. The tag compares the received hash with its version of $H(r_2, k)$. If they match, the tag knows that the reader is authentic.

F. Adding and Removing Tags

In a dynamic system, new tags may be added and existing tags may be removed. To add a new tag, the RFID reader randomly selects a new secret key k' and a new path identifier p', which is broken into d path indices. The reader then inserts a new path into the tree based on the path indices in exactly the same way as it adds a new tag into the system during the initialization phase; see description in Section IV-C. After that, the reader uploads the new key materials to the tag.

To remove a tag, the reader tries to delete the path specified by $p[1]$, $p[2]$, ..., $p[d]$ from the tree. Each node in the tree carries a counter for the number of tags whose paths traverse this node.[1] The reader first removes the leaf node of the tag. It then traverses backward along the path to be removed by using the path indices in the reverse order. When it visits a node, it decreases the node's counter by one and removes the node if its counter becomes zero. When a node is removed, all its children is removed, too.

G. Correctness

First, we prove that an authentic tag will pass authentication. We only need to show that *the RFID reader is able to figure out the correct values of path indices* $p[i]$, $1 \leq i \leq d$, such that it can find the correct leaf node where the secret key shared with the tag is stored. We prove by induction: Suppose the reader has figured out the correct values of $p[j]$, $j < i$. Using these link indices, the reader traverses the tree to a node on the ith level, and the node stores the key $s[i]$ for its children group. The tag uses $H(r, s[i])$ and $p[i]$ to compute f_i. Using the same key $s[i]$, the reader will compute the same hash value $H(r, s[i])$. Since f_i is a unique prefix that the $p[i]$th segment of $H(r, s[i])$ possess, it can be used to identify that segment and its index $p[i]$. Therefore, the reader is able to figure out $p[i]$. The base case for induction is $i = 1$, where the traversal begins from the root and the reader knows $s[1]$, which is the key for the children group of the root.

Second, a counterfeit tag cannot pass authentication. Because a counterfeit tag does not have a shared secret with the reader, it cannot pass the final authentication based on $H(r, k)$ at the end of the decoding phase.

Finally, we need to show that our protocol meets the privacy-preserving requirement, i.e., the authentication data, $\{r_2, f_1, f_2, ..., f_d, H(r, k)\}$, look random each time a tag is authenticated. It is well known that the output of a cryptographic hash function appears totally random to someone who does not know the secret key; intuitively, as the input changes, any bit in the output has 50% chance to change [18]. In fact, hash functions such as SHA-1 have been used to generate random bit sequences [19]. We know that f_i consists of bits taken out from $H(r, s[i])$, $1 \leq i \leq d$. Because the input r to these hashes changes for each authentication, bits in the hash output $H(r, s[i])$ will appear random to anyone who do not know the keys $s[i]$. This is true for any bits in the hash output, including f_i, which is the prefix of the $p[i]$th segment in $H(r, s[i])$. Hence, the privacy-preserving requirement is met.

H. Efficiency

We evaluate protocol efficiency mainly based on two criteria: size of authentication data and amount of computation. First, we determine the expected size of the authentication data that a tag transmits. The authentication data consists of three components: (1) r_2, which has a fixed length, e.g., 64 bits. (2) f_i, $1 \leq i \leq d$, which has a variable length. (3) $H(r, k)$, which has a fixed length, e.g., 160 bits. Let m_i be the length of f_i. We derive the expected value of m_i below.

Let l be a positive integer constant. Consider the prefix of l bits in each segment $v[i]$. The probability for the prefix of $v[i]$ to be different from the prefix of another segment $v[j]$ is $1 - \frac{1}{2^l}$. The probability for the prefix of $v[i]$ to be different from the prefix of any other segment $v[j]$, $0 \leq j < \sigma$, $j \neq i$, is $(1 - \frac{1}{2^l})^{\sigma-1}$. The following statement is obviously true: If we can find a prefix of $v[i]$ that is unique and has a length smaller than l, then we must be able to find a prefix of $v[i]$ that is unique and has a length of l. Thus we have

$$Prob\{m_i \leq l\} = (1 - \frac{1}{2^l})^{\sigma-1}.$$

Because $Prob\{m_i \leq l-1\} = (1 - \frac{1}{2^{l-1}})^{\sigma-1}$, we must have

$$Prob\{m_i = l\} = (1 - \frac{1}{2^l})^{\sigma-1} - (1 - \frac{1}{2^{l-1}})^{\sigma-1}. \quad (1)$$

Let $E(m_i)$ and $Var(m_i)$ be the expected value and variance of m_i, respectively. From (1), we have

$$E(m_i) = \sum_{l=1}^{\infty} l \times Prob\{m_i = l\}, \quad (2)$$

$$\begin{aligned} Var(m_i) &= E(m_i^2) - E(m_i)^2 \\ &= \sum_{l=1}^{\infty} l^2 \times Prob\{m_i = l\} \\ &\quad - (\sum_{l=1}^{\infty} l \times Prob\{m_i = l\})^2. \end{aligned} \quad (3)$$

Figure 4 shows $Prob\{m_i = l\}$ with respect to l when $\sigma = 16$. It quickly approaches to 0 as l increases. $E(m_i)$ is about 5.3 bits, and $Var(m_i)$ is about 3.4 bits. This means that

[1]This counter is initialized to zero. It is increased by one whenever a tag is added into the system and the newly created path for the tag traverses the node.

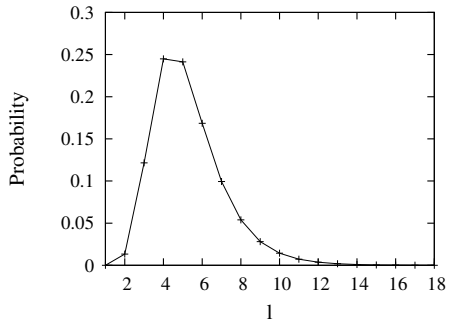


Fig. 4. Probability of $m_i = l$ with respect to l.

CNP only transmits 5.3 bits on average for each hash value $H(r, s[i])$ that would have to be transmitted in its entirety by Dimitriou's protocol.

Let h be the number of bits in a hash output, $|r|$ be the number of bits in r, and denote $E(m_i)$ as e. We know $e \ll h$. The expected size of the authentication data in CNP is $|r|+de+h$. The size of the authentication data in Dimitriou's protocol is $|r| + dh$, and the size of the authentication data in ACTION is $|r| + (d + 1)h$. Suppose $|r| = 64$, $d = 30$, $\sigma = 16$, and $h = 160$. The ratio of the data size in CNP to the data size in Dimitriou's protocol is 0.079 : 1. The raio of the data size in CNP to the data size in ACTION is 0.076 : 1.

Next, we study the computation overhead. The numbers of hash computations by a tag for each authentication are $d+1$ for CNP, d for Dimitriou's protocol, and $d+1$ for ACTION. The amount of computation on the reader differs. For CNP, the reader performs $d+1$ hash computations, exactly the same hashes as the tag does. For Dimitriou's protocol and ACTION, the number of hash computations by the reader is $O(\sigma d)$. In the worse case, Dimitriou's protocol performs σ on each of the d levels in the tree. In the best case, it performs one hash on each level. On average, it needs to perform $(\sigma + 1)/2$ hashes on each of the d levels. If $d = 30$ and $\sigma = 16$, the ratio of the reader's hash computations in CNP to that in Dimitriou's protocol is 0.12 : 1.

I. Variable Length of f_i

f_i has a variable length, which complicates the message format design for authentication data sent from a tag to a reader. A simple solution is to first transmit the lengths of all f_i, $1 \le i \le d$, before transmitting the bits of f_i, $1 \le i \le d$, in concatenation. This solution adds extra bits to the data sent by the tag. Another solution is to design a new cryptographical encoding method that generates authentication data of a fixed length. In the next section, we design an enhanced compact navigation protocol (ECNP) to achieve this goal.

V. Enhanced Compact Navigation Protocol

This section presents the enhanced compact navigation protocol (ECNP). It also consists of three phases: *initialization phase, encoding phase*, and *decoding phase*. The initialization phase is the same as what CNP does. ECNP differs from CNP in the way it computes authentication data in the encoding phase and the way it recovers the path indices in the decoding phase. Therefore, we only describe the encoding phase and the decoding phase below.

A. Encoding Phase

At the beginning of the encoding phase, the reader sends a random number r_1 to a tag. The tag produces another random number r_2 and XORs it with r_1 to produce r. After that, it starts to encode its path indices, $p[1]$, ..., $p[d]$.

To encode $p[1]$, the tag computes a hash value $H(r, s[1])$ and splits it into σ segments of equal length, denoted as $v[0], v[1], ..., v[\sigma-1]$. The tag then sorts $v[0], v[1], ..., v[\sigma-1]$ in ascending order. The result is a sorted list, $v'[0] \le v'[1] \le ... \le v'[\sigma - 1]$. It uses the index of $v[p[1]]$ in the sorted list, denoted as idx_1, to encode $p[1]$. Namely, if $v[p[1]] = v'[j]$, then $idx_1 = j$, which specifies how many other segments are smaller than $v[p[1]]$. The tag will transmit j to the RFID reader. Because $0 \le idx_1 < \sigma$, it has a fixed length of $\log_2 \sigma$ bits. When $\sigma = 16$, it is 4 bits long.

We use an example to illustrate the above idea. Suppose $\sigma = 4$, $H(r, s[1])$ = "1100||0000||1111||0001", and $p[1] = 1$. We have

$$v[0] = \text{“1100”}$$
$$v[1] = \text{“0000”},$$
$$v[2] = \text{“1111”},$$
$$v[3] = \text{“0001”}.$$

$v[p[1]] = v[1] =$ "0000". After sorting, we have

$$v'[0] = v[1] = \text{“0000”}$$
$$v'[1] = v[3] = \text{“0001”},$$
$$v'[2] = v[0] = \text{“1100”},$$
$$v'[3] = v[2] = \text{“1111”}.$$

Because $v[p[1]] = v'[0]$, the tag uses 0 to encode $p[1]$. Hence, $idx_1 = 0$.

What about $v[p[1]]$ is not unique, i.e., it is equal to another segment? In CNP, the tag has to compute more hash values and make $v[p[1]]$ longer in order to make it unique. This is unnecessary in ECNP. We use the previous example but change $H(r, s[1])$ to "1100||0000||1111||0000". We have

$$v[0] = \text{“1100”}$$
$$v[1] = \text{“0000”},$$
$$v[2] = \text{“1111”},$$
$$v[3] = \text{“0000”}.$$

Still, $v[p[1]] = v[1] =$ "0000". If we use bobble sort, we have

$$v'[0] = v[1] = \text{“0000”}$$
$$v'[1] = v[3] = \text{“0000”},$$
$$v'[2] = v[0] = \text{“1100”},$$
$$v'[3] = v[2] = \text{“1111”}.$$

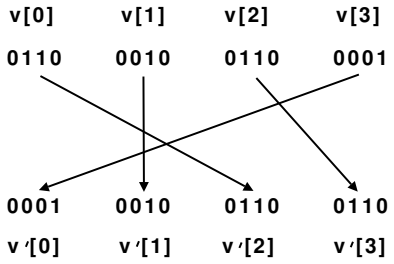


Fig. 5. Relationship between the original segment list $v[i]$, $1 \leq i \leq \sigma$, and the sorted list $v'[i]$, $1 \leq i \leq \sigma$.

Notice that bobble sort does not change the relative positions of $v[1]$ and $v[3]$. Before sorting, $v[0]$ is ahead of $v[3]$ in the list. After sorting, $v'[0] = v[1]$ is still ahead of $v'[1] = v[3]$. This helps us determine that $v[p[1]]$, which is $v[1]$ in the original list, is mapped to $v'[0]$ instead of $v'[1]$, and therefore idx_1 is 0, not 1.

In general, suppose k segments of $H(r, s[1])$ has the same value of $v[p[1]]$. Among these segments, suppose k' is ahead of $v[p[1]]$ in the list before sorting. Then after sorting, we select the $(k'+1)$th segment whose value is $v[p[1]]$, and use its index in the sorted list as the value of idx_1.

Similarly, the tag can compute idx_i, $1 < i \leq d$, from $H(r, s[i])$ to encode $p[i]$. It sends $\{r_2,\ idx_1,\ idx_2,...,idx_d,\ H(r,k)\}$ as authentication data to the reader.

B. Decoding Phase

After the reader receives the authentication data $\{r_2, idx_1, idx_2, ...,\ idx_d,\ H(r,k)\}$ from the tag, it first reproduces r by XORing r_2 with r_1. The reader then decodes idx_i, $1 \leq i \leq d$, to recover the link indices $p[i]$, based on which it can traverse the tree to find the leaf node carrying the secret key k of the tag. Below we only explain how to decode idx_1 for $p[1]$. Other $p[i]$ values can be similarly derived.

To decode idx_1, the reader repeats the same process as the tag does in the previous phase. It computes the hash value $H(r, s[1])$, divides it into σ segments, $v[0], v[1], ..., v[\sigma-1]$, and then sorts them in ascending order by bobble sort. The sorted list is denoted as $v'[0], v'[1], ..., v'[\sigma-1]$. Suppose $v[j]$ in the original segment list is mapped to $v'[idx_1]$ in the sorted list. The reader knows that $p[1] = j$. Figure 3 shows the relationship between an original segment list and its sorted list. In this example, suppose $idx_1 = 3$. The mapping between the two lists shows unambiguously that $v[2]$ is mapped to $v'[idx_1]$. Hence, $p[1] = 2$.

After the reader decodes idx_i, $1 \leq i \leq d$, and finds all path indices, it uses these indices to traverse the tree and reach a leaf node that carries a key k'. It performs a keyed hash $H(r, k')$, and compares it with the received $H(r, k)$. If they are the same, the reader knows that $k' = k$ and the tag is authentic. To allow the tag to authenticate the reader, the reader computes $H(r_2, k)$ and transmits it to the tag. The tag compares the received hash with its version of $H(r_2, k)$. If they match, the tag knows that the reader is authentic.

C. Correctness

First, we prove that an authentic tag will pass authentication in ECNP. Similar to Section IV-G, we only need to show that *the RFID reader is able to figure out the correct values of path indices* $p[i]$, $1 \leq i \leq d$. We prove by induction: Suppose the reader has figured out the correct values of $p[j]$, $j < i$. Using these link indices, the reader traverses the tree to a node on the ith level, and the node stores the key $s[i]$ for its children group. The tag computes $H(r, s[i])$, divides it into σ segments, identifies the $p[i]$th segment, and then uses the index location idx_i of this segment in the sorted order as encoded information, which is transmitted to the reader. Using the same key $s[i]$, the reader will compute the same hash value $H(r, s[i])$, divide it into the same set of segments, and sort them. The reader finds the idx_ith segment in ascending order, and maps it back to the original order to find its location $p[i]$ there. Since the mapping of segments in the original order to the ascending order is one-to-one, there is no ambiguity in the process of recovering $p[i]$. The base case for induction is $i = 1$, where the traversal begins from the root and the reader knows $s[1]$, which is the key for the children group of the root.

Second, a counterfeit tag cannot pass authentication because it does not have a shared secret k with the reader, which is needed to pass the authentication based on $H(r, k)$.

Finally, we show that our protocol meets the privacy-preserving requirement, i.e., the authentication data, $\{r_2, idx_1, idx_2, ..., idx_d, H(r,k)\}$, look random each time a tag is authenticated. We know that, for any input change, the output of a cryptographic hash function will appear to change randomly: Each bit has a chance of 50% of flip, when observed by someone who does not have the key [18]. Because the input r changes for each authentication, bits in the hash output $H(r, s[i])$, $1 \leq i \leq d$, will change randomly. Since bits in all σ segments of $H(r, s[i])$ change arbitrarily and no segment is more special than any other, the $p[i]$th segment may appear at any location in the sorted order. It may be the largest, the smallest, or in any middle location, with equal probabilities. Hence, idx_i (the location index of the $p[i]$th segment in the sorted order) will appear to be random in the range of $[0, \sigma)$. That is, the authentication data look random each time the tag is authenticated. The privacy-preserving requirement is met.

D. Efficiency

The size of the authentication data in ECNP is $|r| + d\log_2\sigma + h$, where h is the size of a hash output. Recall that the expected size of the authentication data in CNP is $|r| + de + h$, plus d values that specify the lengths of f_i, $1 \leq i \leq d$. Note that e is the expected length of f_i. When $\sigma = 16$, $e = 5.3$. The size of the authentication data in Dimitriou's protocol is $|r| + dh$, and the size of the

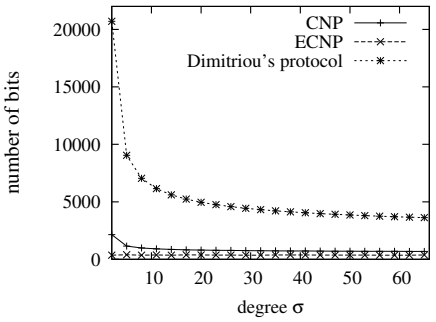


Fig. 6. Size of authentication data transmitted by a tag in each authentication with respect to σ.

authentication data in ACTION is $|r| + (d+1)h$. Suppose $|r| = 64$, $d = 30$, $\sigma = 16$, and $h = 160$. The ratio of the data size in ECNP to the data size in Dimitriou's protocol is 0.071 : 1. The ratio of the data size in ECNP to the data size in ACTION is 0.068 : 1.

In the following, we give another numerical comparison with respect to σ. Suppose the length of the path indicator p is 160 bits, the length of a hash output is 160, and the length of each random number is 64 bits. Each link index is $\log_2 \sigma$ bits long. Thus, the depth d of the tree is set to $\frac{160}{\log_2 \sigma}$.

In Dimitriou's protocol, a tag sends its random number r_2 and d hash values. The total length of its authentication data is

$$64 + d \times 160 = 64 + \frac{25600}{\log_2 \sigma}, \text{ as } d = \frac{160}{\log_2 \sigma}. \quad (4)$$

CNP requires a tag to transmit the authentication data $\{r_2, f_1, f_2, ..., f_d, H(r,k)\}$ with the total length of

$$64 + \sum_{i=1}^{d}(m_i + 10) + 160 = 224 + \sum_{i=1}^{160/\log_2 \sigma} m_i + \frac{1600}{\log_2 \sigma}, \quad (5)$$

where m_i is the length of f_i and we assume 10 bits are sent with f_i to specify its length. We have discussed the relationship between $E(m_i)$ and σ in (1) and (2).

In ECNP, a tag needs to transmit the authentication data $\{r_2, idx_1, idx_2, ..., idx_d, H(r,k)\}$, whose total length is

$$64 + d\log_2 \sigma + 160 = 384, \text{ as } d = \frac{160}{\log_2 \sigma}. \quad (6)$$

We vary σ from 2 to 64. Figure 6 shows the number of authentication bits transmitted by a tag in one authentication. The number of bits decreases when σ increases for all three protocols. Both CNP and ECNP require much smaller numbers of bits than Dimitriou's protocol. Even when σ is 64, the difference remains significant.

The numbers of hash computations in ECNP is the same as that in CNP. See Section IV-H for analysis.

VI. Conclusion

This paper designs two privacy-preserving protocols based on a new technique, called cryptographical encoding, which can significantly reduce the authentication data transmitted by each tag and computation overhead at the reader. The analysis shows that the new protocols are able to reduce authentication data by more than an order of magnitude and computational demand by about an order of magnitude, when comparing with the best existing protocol.

VII. Acknowledgements

This work was supported in part by the US National Science Foundation under grant CPS-0931969. We would also like to thank the anonymous reviewers for their constructive comments.

References

[1] T. Kriplean, E. Welbourne, N. Khoussainova, V. Rastogi, M. Balazinska, G. Borriello, T. Kohno, and D. Suciu, "Physical Access Control for Captured RFID Data," *IEEE Pervasive Computing*, 2007.

[2] Y. Liu, L. Chen, J. Pei, Q. Chen, and Y. Zhao, "Mining Frequent Trajectory Patterns for Activity Monitoring Using Radio Frequency Tag Arrays," *Proc. of IEEE PerCom*, 2007.

[3] L. Ni, Y. Liu, and Y. C. Lau, "Landmarc: Indoor Location Sensing using Active RFID," *Proc. of IEEE PERCOM*, 2003.

[4] Y. Li and X. Ding, "Protecting RFID Communications in Supply Chains," *Proc. of ASIACCS*, 2007.

[5] B. Sheng, C. Tan, Q. Li, and W. Mao, "Finding Popular Categories for RFID Tags," *Proc. of ACM Mobihoc*, 2008.

[6] Chiu C. Tan, Bo Sheng, and Qun Li, "How to monitor for missing RFID tags," *Proc. of IEEE ICDCS*, 2008.

[7] C. Wang, H. Wu, and N. F. Tzeng, "RFID-based 3-D Positioning Schemes," *Proc. IEEE INFOCOM*, 2007.

[8] C. H. Lee and C. W. Chung, "Efficient Storage Scheme and Query Processing for Supply Chain Management Using RFID," *Proc. ACM SIGMOD*, 2008.

[9] A. Nemmaluri, M. Corner, and P. Shenoy, "Sherlock: Automatically Locating Objects for Humans," *Proc. of ACM MobiSys*, 2008.

[10] L. Ravindranath, V. Padmanabhan, and P. Agrawal, "Sixthsense: RFID-based Enterprise Intelligence," *Proc. of ACM MobiSys*, 2008.

[11] S. Weis, S. Sarma, R. Rivest, and D. Engels, "Security and Privacy Aspects of Low-cost Radio Frequency Identification Systems," *Lecture notes in Computer Science*, 2004.

[12] Q. Yao, Y. Qi, J. Han, J. Zhao, X. Li, and Y. Liu, "Randomizing RFID Private Authentication," *Proc. of IEEE PERCOM*, 2009.

[13] Tassos Dimitriou, "A Secure and Efficient RFID Protocol that could make Big Brother (partially) Obsolete," *Proc. of IEEE PERCOM*, 2006.

[14] L. Lu, J. Han, L. Hu, Y. Liu, and L. Ni, "Dynamic Key-Updating: Privacy-Preserving Authentication for RFID Systems," *Proc. IEEE PERCOM*, 2007.

[15] L. Lu, J. Han, R. Xiao, and Y. Liu, "ACTION: Breaking the Privacy Barrier for RFID Systems," *Proc. of IEEE INFOCOM*, 2009.

[16] U. Erlingsson, M. Manasse, and F. McSherry, "A Cool and Practical Alternative to Traditional Hahs Tables," *Proc. of WDAS*, 2006.

[17] L. Lu, Y. Liu, and X. Li, "Refresh: Weak Privacy Model for RFID Systems," *Proc. of IEEE INFOCOM*, 2010.

[18] C. Kaufman, R. Perlman, and M. Speciner, "Network Security: Private Communication in a Public World," *Prentice Hall; 2 edition*, 2002.

[19] "SHA1 Random Number Generation," *https://developer-content.emc.com/docs/rsashare/share_for_java/1.1/dev_guide/group__JCESAMPLES__RNG__SHA1.html.*